Lecture 35

Threads

Reading for next time: Big Java 21.4

What is a Thread?

- **Imagine a Java program that is reading large files over the Internet from several different servers (or getting data from several sensors over a local area network or…) . Some of these servers might be under heavy load or on slow connections to the Internet. Others might return the data quickly.**
- **Most of the time in our program will be spent waiting for input over the network. One programming approach is straightforward:**

  read file 1 from server A
  read file 2 from server B
  read file 3 from server C
  . . . .

What is a Thread?, 2

- **Doing these reads sequentially is inefficient since the loading of file 2 from server B wouldn't start until all of file 1 is loaded.**
- **A much faster approach is to start reading from each file concurrently, and handle the partial files as they arrive.**
- **This requires the ability to have several tasks proceeding in parallel (as though they were each assigned to a separate independent processor).**

What is a Thread?, 3

- **Most computers still only have a single processor, so what we really want is an easy way for the program to switch among arriving data sources.**
- **More generally, we would like to be able to write programs where the "flow of control" branches , and the branches proceed in parallel.**
- **The processor can achieve this by switching between the different branches of the program in small time increments.**
- **This is the strategy behind *threads*.**

Threads vs. Processes

- **Most operating systems allow concurrent *processes* to proceed in parallel.**
- ***Processes* are expensive but safe. Processes are so well insulated from each other that it is both complex and often expensive to communicate between them.**
- ***Threads* are cheap, but different threads running in the same process are not well-insulated from each other.**

Java Support for Threads

- **Java is the only widely used language where the support for threads is a part of the language.**
- **Ada, a language developed by the Department of Defense, also has built-in support for threads, but Ada is little used outside DoD contexts.**
- **In other languages such as C and C++, there are libraries to implement threads which are more or less standardized.**

Java is Inherently Multithreaded

- In Java, the garbage collection of unreferenced objects is performed by the Java runtime system in a separate thread.
- Java also uses a separate thread to deliver user interface events. This allows a program to remain responsive even while it is involved in a long running calculation or I/O operation.
- Think how you would implement a "Cancel" function if you could not use threads.
- This means Java is inherently multithreaded. The Java runtime environment uses multiple threads even if the user's program doesn't.
- But programmers can also use threads in their own code. Our multiple file download strategy requires threads.

Simple Thread Example

- In this example we will implement a multithreaded download program
- The program uses a separate `Thread` to read each URL from a Web server on the Internet and copy the contents of that URL to a local file.
- We call the class that does the work and extends the `Thread` class, `URLCopyThread`.
- `URLCopyThreadMain` creates a new instance of `URLCopyThread` for each copy operation.

URLCopyThreadTest

```
public class URLCopyThreadTest {
  public static void main(String argv[]) {
    String[][] fileList = {
      {"server","home.html"},
    {"http://www.pepysdiary.com/", "index.html"},
    {"http://microscopy.fsu.edu/micro/gallery/dinosaur/dino1.jpg",
     "dino1.jpg"},
    {"http://www.boston.com/","globe.html"},
    {"http://java.sun.com/docs/books/tutorial/index.html",
     "tutorial.index.html"},
    };  // End string array
```

URLCopyThreadMain, 2

```
    for (int i=0; i<fileList.length; i++) {
      Thread th;
      String threadName = new String( "T" + i );
      th = new URLCopyThread( threadName,
                              fileList[i][0],
                              fileList[i][1] );
      th.start();
      System.out.println("Thread " + th.getName() +
       " to copy from " + fileList[i][0] + " to " +
       fileList[i][1] + " started" );
    }
  }
}
```

URLCopyThread

```
import java.io.*;
import java.net.*;
public class URLCopyThread extends Thread {
  private URL fromURL;
  private BufferedInputStream input;
  private BufferedOutputStream output;
  private String from, to;

  public URLCopyThread(String n, String f, String t) {
    super( n );
    from = f;
    to = t;
    try {
      fromURL = new URL(from);
      input = new BufferedInputStream( fromURL.openStream());
      output = new BufferedOutputStream( new FileOutputStream(to));
    }
    catch(MalformedURLException m) {
      System.err.println("MalformedURLException "+ from); }
    catch(IOException io) {
      System.err.println("IOException " + io); }
  }
```

URLCopyThread, 2

```
  public void run() {
    byte [] buf = new byte[ 512 ];
    int nread;
    try {
      while((nread=input.read(buf,0,512)) > 0) {
        output.write(buf, 0, nread);
        System.out.println( getName() + ": " +
          nread + " bytes" );
    }
    input.close();
    output.close();
    System.out.println("Thread " + getName() +
        " copying " + from + " to " + to + "finished");
    }
    catch(IOException ioe) {
      System.out.println("IOException:" + ioe);
    }
  } // end of run() method
} // end of URLCopyThread class
```

Exercise

- **Download and run URLCopyThreadTest and URLCopyThread**
  - **Which threads start first?**
  - **Which overlap?**
  - **If you run it multiple times, do you get different runtime behavior?**
  - **Switch the order of the URLs in the program and run the program. What happens?**

How Do I Tell a Thread What to Do?

There are two approaches:

1. **Your class can inherit from the Thread class and override its method `public void run()`.**

```
public class MyThread extends Thread  {
    public void run() {
    // code executed in the Thread goes here
    }
}
```

In main(), for example, you create an instance of your thread like this:

```
Thread t = new MyThread();
```

- **After the thread object t is created, its run() method is executed by calling start(); it's just like main() in an overall program: run() means "start here"**

How Do I Tell a Thread What to Do?, 2

2. You can write a separate class that implements the `Runnable` interface, which contains only a single method:

```
public interface Runnable {
  public void run();
}
```

   You create the `Thread` by using your `Runnable` object as a constructor argument (see next slide)

   One reason to use this approach is that Java classes can only inherit from a single class. If you want to define a thread's `run()` method in a class that already inherits from another class, you cannot use the first strategy.

Runnable Example

For example, consider the class `JFrameInThread` defined as

```
public class JFrameInThread
  extends JFrame implements Runnable {
    // constructors and other methods go here
    public void run() {
      // code executed in the Thread goes here
    }
}
```

If we wanted an instance of the `JFrameInThread` class to run in its own `Thread`, we could use the statement

```
Thread t = new Thread(new JFrameInThread() );
```

Starting and Stopping Threads

- ***How do you start a thread working:*** **Call `start()` on the thread instance.**

```
Thread t = new MyThread();
t.start();  // Invokes run() for thread,
            // which is like a main() for it
```

- ***How do you stop a thread and destroy it:*** **Let the `run()` method complete and the thread reference go out of scope or set the reference to null. The garbage collector will reclaim the thread's storage.**
  - **`t.stop()` is deprecated.**
  - **`t.stop()` could stop a thread halfway through a method call and leave the program in an inconsistent state. (It's like killing a regular program at some arbitrary point.)**

How to Tell If a Thread is Still Running

- **You can ask it:**

```
Thread t = new MyThread();
t.start();
. . .
if ( t.isAlive() ) // it's still running
else               // it isn't
```

- **Or you can wait for it:**

```
t.join();  // blocks until t completes
```

RandomExample

- **Download RandomExample and run it**
  - **It runs the Java random number generator 10,000,000 times each time you enter a number, and tells you how often that number came up**
    - **This is not very useful, of course. It's just a stand-in for some computational method that takes a long time to run**
  - **Notice that you have to wait a while for the results, before the JOptionPane is ready for input again**
  - **We'll put the computations into threads so that control returns to the GUI as soon as possible**
  - **RandomTest.java contains a package access class, RunRandom for convenience**
    - **We'll have 3 versions of all this at the end**

RandomExample, p.1

```
import java.util.*;
import javax.swing.*;

class RunRandom {
   private static final int SIZE = 10000000;
   private int lookup;

   public RunRandom(int i) {
        lookup = i;
   }

   public void findOccurrences() {
        Random r = new Random();
        int count = 0;
        for (int i = 0; i < SIZE; i++)
               if (lookup == r.nextInt(255))
                      count++;
        System.out.println("Your number " + lookup +
                " was found " + count + " times.");
   }
}
```

RandomExample, p.2

```
public class RandomExample {
   public static void main(String[] args) {
     int choice = 0;
     while (true) {
        String num = JOptionPane.showInputDialog(
            "Enter a number to search [0-255, -1 to quit]: ");
        choice = Integer.parseInt(num);
        if (choice == -1)
          break;
        RunRandom r = new RunRandom(choice);
        r.findOccurrences();
     }
     System.out.println("Done");
     System.exit(0);
   }
}
```

Exercise

- **Modify the program to create a thread each time a number is entered:**
- **In class RunRandom:**
  - **Have RunRandom extend Thread**
  - **Give it a public void run() method**
    - **The method body is like a 'main()' for the thread. It tells it what to do when it is started. Write the method body; it's very short!**
  - **No other changes in RunRandom**
- **In class RandomExample, in main method:**
  - **Create an array to hold RunRandom objects. Size 10 or so is fine**
  - **Each time the user picks a number, create a new RunRandom object and start it**
  - **Let the threads run until completion**
  - **Remove the System.exit(0) call**

Exercise, part 2

- Experiment and observe:
  - Where does 'Done' appear in the output? Why?
  - Put the System.exit(0) call back in and see what happens. Why?
  - But, without System.exit(0) and with a JOptionPane, your program doesn't terminate correctly.
- A final change:
  - In main, after the while loop, add a try-catch block:

```
try {
    rArray[--i].join();  // Join (wait for) last thread
    System.out.println("Try block:"+ i);
    System.exit(0);
} catch (InterruptedException e) {
    System.exit(1);
}
```

  - This isn't really enough, since the threads could finish out of order, but it gives an introduction to the issue
  - A better approach is to loop thru all threads using `t.isAlive()` until none are

Exercise: Runnable

- Modify the program you just wrote to implement Runnable rather than extend Thread:
- In class RunRandom:
  - Have RunRandom implement Runnable instead of extending Thread.
  - No super() call in constructor, if you had one.
  - The public void run() method is unchanged
  - No other changes in RunRandom
- In class RandomExample, in main method:
  - Create an array to hold Thread objects, *not RunRandom*
  - Each time the user picks a number, create a new *Thread* object, *using a new RunRandom object in its constructor*, and start it
    - See the previous slide on Runnable
  - No other changes in main()